Project report – Air filter classification

1) Task

Develop multiple CCNs and achieve the best possible classification accuracy on given dataset of air filters. Given the “simplicity” of the task do many experiments with augmentation and training hyperparamenters. Show systematic approaches.

2) Existing solutions

The task itself is not very difficult. In the company where I work as technical planning trainee, the task is solved with an old camera. The camera has built conventional algorithms for machine vision. It recognizes one of 11 types of filters and sends signal to PLC which then proceeds to communicate with robot to handle the following steps. With changing conditions in the factory, the camera solution is not as robust as it should be. The accuracy of classification sometimes drops as low as 85 %, which then leads to additional scrap.

The new presented solution from external company also used conventional algorithms too but no deep learning. I was wondering if I could develop CNN which would surpass the current camera and would be a much cheaper solution than the one proposed.

From various reasons my company decided to go with the external company. I was however allowed to use the dataset for my experiments. So, I decided to try it although I study mechanical engineering and have no experience with neural networks whatsoever.

3) My solution

Dataset

At the beginning I had 11 classes (attachment classes shown), each of which had at least 500 grayscale pictures, which is not a lot for CNN to train. First, I manually went through the data and selected 500 images from each class. That way I made sure no class was over- or underrepresented and that the data were diversified. Then I cropped all images to be square and 224x224 pixels.

With the lack of data, no model will work as it should. I knew this step was very important for the model. I wanted to know how model performs without augmentation though.

When I started a didn’t apply any augmentation on training and validation data (80/20 split). However, I did apply very light augmentation on the testing data. The model l quickly converged towards 99,98 % mark in training as well as in validation. On the augmented testing data it failed completely, it couldn’t figure out just a simple noise, brightness and contrast variations and achieved accuracy around 20 %.

Before the next step I was thinking how to deal with the augmentation situation with only 500 images of each class. I made the decision to augment from the same data for training and testing and make sure the training and testing data were increasingly different and more difficult for model to recognize. I am aware that this

The next step was to build sufficient training data. I wanted to apply TensorFlow augmentation directly in the training script, but after some attempts I realized that some of the augmented pictures

were completely black. I think it had to do something with the normalization and the fact that I had to fake the RGB channels of the pictures for the network to be able to work with them. So, I have decided to completely avoid TF augmentation and went on with my own in cv2. I made no original data included in augmentation as I planned to use them for testing (on easiest level). I created Three scripts. First, I made 3000 images per class with rotation, random brightness, contrast, zoom and crop. I realized that brightness and contrast amplifications hurt my model a lot, because classes 8,9,10,11 became indifferentiable even for my human eye. Then was created larger dataset of 5000 images per class without brightness and contrast amplification, after which I added few more adjustments like Gaussian blur and salt and pepper noise. The last dataset was then used for proper systematic training of model.

For testing I created three datasets. First was the original one without any augmentation. The second had decently difficult augmentation and the third one was meant to be the most challenging as the data were distorted.

Training, validation and testing

I did a lot of testing on my own, because I have never trained any network and needed to learn basics first. But for the final training session I used just the last dataset with the most diverse images.

For training I used Tensorflow and Keras. I used pretrained networks MobileNetV2 and EfficientNetB0, which are both lighter models with under 10 million parameters, but EfficientNetB0 is a bit more complex and should be more accurate. I used 80/20 training and validation split. I tried to fine-tune those models for the best performance. I took a systematic approach: I trained 28 networks with different combinations of hyperparameters.

Pretrained model: MobileNetV2 a EfficientNetB0; Nr. of trainable layers: 0, 5, 10, 15; learning rate: 1E-4, 1E-2; regularization: dropout 0.2, dropout 0.4 + L2 on the last two layers. Then I used the three testing datasets for evaluation. I expected testing the original to be very precise: 95% +, on the other two I didn't know what to expect. After testing I decided to disregard all models with learning rate 1E-2, because models were disaster as I should have anticipated. They were so overfitted, they all achieved below 30% in testing.

I made the testing part deliberately difficult because it is true that the project itself is not very high-level stuff and there is not enough data. For the network to perform well on the original dataset, there is not much to go wrong. I wanted to challenge the model and myself as well.

For evaluation of the testing, I added confusion matrix to the code. It plots CM after each of three tests of the model. I have noticed that Filter type 9 a 10 are often confused, because they have the exact same shape, and they only differ in the strap of glue in the middle. The strap then disappears if the contrast or brightness is too high.

Evaluation

The table below shows accuracy of each model. Model configuration **Network_trainable-layers_learinig-rate_regularization(Y/N)**

Network	Train. / val.	Test original	Test medium	Test difficult
MobileNetV2_0_1E-4_N	0.997/0.996	0.995	0.874	0.864
MobileNetV2_5_1E-4_N	0.990/0.904	0.746	0.670	0.592
MobileNetV2_10_1E-4_N	0.991/0.980	0.977	0.779	0.727
MobileNetV2_15_1E-4_N	0.991/0.948	0.958	0.780	0.720
MobileNetV2_0_1E-5_Y	0.999/0.998	0.999	0.922	0.865
MobileNetV2_5_1E-5_Y	0.9416/0.9533	0.889	0.743	0.662
MobileNetV2_10_1E-5_Y	0.948/0.967	0.928	0.740	0.689
MobileNetV2_15_1E-5_Y	0.975/0.979	0.959	0.800	0.736
EffitientNetB0_0_1E-4_N	0.999/0.999	0.999	0.934	0.935
EffitientNetB0_5_1E-4_N	0.998/0.998	1.000	0.962	0.882
EffitientNetB0_10_1E-4_N	0.998/0.999	1.000	0.966	0.844
EffitientNetB0_15_1E-4_N	0.998/0.998	1.000	0.963	0.881
EffitientNetB0_0_1E-5_N	0.998/0.998	0.999	0.926	0.889
EffitientNetB0_0_1E-5_Y	0.998/0.999	0.999	0.900	0.911
EffitientNetB0_5_1E-5_Y	0.991/0.998	0.993	0.950	0.839
EffitientNetB0_10_1E-5_Y	0.993/0.998	0.992	0.944	0.834
EffitientNetB0_15_1E-5_Y	0.993/0.998	0.993	0.953	0.863

Table 1 Performance of each model

Conclusion

I have tried to develop the most accurate classifier I could and also show some systematic approach in augmentation as well as training.

After the training it's clear that fine tuning didn't bring much improvement to accuracy – dataset is too little and too simple. However, unlocking a couple of layers really shortened the training time of the model. The learning rate 1E-5 and regularization led to similar results. But I also tested one model without regularization with 1E-5 learning rate and the performance was worse than with 1E-4 learning rate without regularization. I noticed models had higher validation accuracy than training accuracy. It makes sense, there is a strong data reduction during training. Maybe if the dropout was less aggressive it could achieve even better results.

Models always performed best at normal dataset which is closest to real world application and therefore achieved a desired goal. They even achieved decent performances in augmented data, which are very unlikely to occur in the real world. As long as there is no overexposed picture which eliminates important features, the model should perform very well.

Models also tend to be more accurate at medium or at difficult augmentation. The more one accuracy increased the more the second decreased. Like in the case of EffitientNetB0_5_1E-4_N and EffitientNetB0_10_1E-4_N, where those models with unfrozen layers showed better performance on medium test than the best model **EffitientNetB0_0_1E-4_N** (which however excelled at difficult test).